Source Code Analysis Tools - Business Case

Christoph Michael, Cigital, Inc. [vita[3]] Steven R. Lavenhar, Cigital, Inc. [vita[4]]

2005-09-28

Organizations are often reluctant to build security into software from the start of the development process. This reluctance stems from a belief that secure programming would increase costs or cause delays. However, experience does not seem to bear out this belief. This article presents a case for using security analyzers.

Even in the most meticulous development process, security bugs can slip into the source code. This is especially true for systems that use legacy software. Defense in depth demands audits of both old and new software to help ensure that no security problems slip through the cracks.

Security scanners automate some of more repetitive and tedious tasks of source-code security auditing. Their strength is finding certain bad programming practices that may lead to security vulnerabilities. With many security analyzers, the rule of thumb is that the easier it would be for an expert to find a security bug, the easier it is for the security analyzer to find it as well. But the purpose of security analyzers is not to be *smarter* than human experts but to be *faster*.

With advances in technology, security analyzers have progressed in two directions. Some analyzers specialize in *software semantics*, trying to predict software behavior through static analysis. Their focus is reducing false alarms and perhaps even finding some problems that would be harder for a human analyst to see. There are also security analyzers that position themselves as style-checkers, and they focus on detecting programming practices that should be avoided in an ultra-paranoid software development setting.

The Case for Software Security Analysis

Management priorities on a software development project are generally *schedule, costs, features,* and *software quality*, in that order. Since schedules tend to be overoptimistic, software quality often gets dropped, along with other functionality that is invisible to an ordinary user. This also includes software security.

Organizations are often reluctant to build security into software from the start of the development process. This reluctance stems from a belief that secure programming would increase costs or cause delays. However, experience does not seem to bear out this belief. In general, it seems that organizations with a more mature development process also produce software faster and more cheaply. While past studies have not specifically focused on software security, there is reason to believe that the same generalization holds true in this area. For example, rigorous type and style checking often points out possible design improvements that are easiest to incorporate while the software product is still in the early stages of design.

Static analysis tools support a secure programming effort by finding and cataloging a large number of *potential* security bugs. Many times these bugs would be easily spotted by a human auditor, but an analysis tool makes the process much faster and more systematic. Once this information is available, it can be used to prioritize risks and thus drive the code-oriented aspects of secure software development.

3. daisy:251 (Michael, C. C.)

4. daisy:197 (Lavenhar, Steven)

Static analysis tools identify potential security bugs during the coding process which can result in cost reductions over the lifetime of the system. According to NIST, the relative cost of repairing software defects increases the longer it takes to identify the bug [NIST 02]. For example, NIST estimates that it can cost thirty times more to fix a coding problem that is discovered after the product has been released than it would have cost if the problem was discovered during unit test or coding.

When to Use Security Analyzers

Security analyzers are frequently used during source-code audits and walkthroughs. Additionally, most security scanners can be used by software developers, so they can be integrated into the development environment. Security analyzers can also make security analysts more efficient.

Like any quality-assurance mechanism, security scanners should be employed as early as possible, since problems found earlier are usually easier and cheaper to fix. On the other hand, the limitations of security analyzers also have to be kept in mind. First, security scanners are not intended to detect design- or architecture-level flaws; they are meant to work on source code. Secondly, security scanners are not currently well suited for finding integration bugs, so nothing much is gained by putting off their use until late in the software development life cycle.

Some security scanners are meant to be used as coding-style checkers, but to be efficient in this role they must be used from the very start of the development phase. Otherwise, it will probably be too expensive to bring the software into compliance with whatever guidelines the analyzer tries to enforce.

One the other hand, those analyzers that try to deduce software behavior usually emphasize technology that reduces false alarms (i.e., apparent vulnerabilities that actually turn out to be unexploitable). While not all tools succeed on this count, reducing false alarms also reduces the amount of work that has to be done to fix the problems an analyzer finds. While it is not a good practice to postpone security analysis, such analyzers might still be useful situations where security analysis *cannot* start during the early development stages, such as in cases where legacy code has to be audited.

Security analyzers can also be used as “badness-ometers” [McGraw 04], which give an organization some idea of how bad the security of their software actually is. In principle, security analyzers can be used this way at any time as long as source code is available, but sooner is better if the organization still hopes to address the security problems that are uncovered during this process.

Required Skills

Most security analyzers are meant to be used by software developers or others with comparable skills or experience. Nonetheless, most commercial analyzers also have advanced features that make them more still more powerful in the hands of software security analysts or persons skilled in source-code analysis.

Potential Benefits

Security analyzers have the following potential benefits:

- **reducing costs over the system lifetime**. Security vulnerabilities identified early in the life cycle are cheaper to fix. Static code scanning tools are used to identify potential security vulnerabilities in the development phase.
- **educating developers about secure programming.** The list of insecure programming practices is long and continues to grow each month. This makes it hard for developers to keep track, especially since most developers are not trained in secure programming. A good security scanner not only finds problems but explains what is wrong and how to fix it. This provides developers with hands-on

feedback on how to improve their programming practices.

- **rechecking legacy code.** Even if the legacy code was developed with security in mind, it is inevitable that new classes of vulnerabilities will have come to light since the code was developed. Security analyzers can help weed out such problems.
- **use as badness-ometers.** Security scanners can help measure how bad software is, though there are a number of reasons that they cannot be used to measure how *good* the software is.
- **automating repetitive and tedious aspects of source-code security audits,** freeing up human security analysts to track down more difficult problems.
- **checking for good programming style from the security standpoint.** Here, however, it must be kept in mind that the analyzer will check for *its own* idea of what constitutes good style unless it is customized. Customization can be time consuming and requires a certain amount of expertise.

What Security Analyzers Can't Do

As mentioned above, security analyzers are not designed to find architecture- or design-level flaws, and they are not well suited for finding integration bugs. Security scanners are also somewhat limited when it comes to analyzing large systems, such as those consisting of many executable components or heterogeneous layers.

It should also be emphasized that, just like human auditors, security analyzers do not find *all* vulnerabilities in a software system, they find only some. Out of all the vulnerabilities present in a given piece of software, a security scanner will be able to find only a certain subset, and that subset will never change unless the scanner is reconfigured or updated with a newer version. After all, the analyzer is just a software application itself. The remaining problems have to be found by human analysts.

It is therefore important not to rely on security scanners as the sole means of or securing software source code. Developers will learn to avoid the kinds of problems that are pointed out by the security scanner, and while this is a good thing, it does not mean that they are also getting better at avoiding other types of security problems that the analyzer *cannot* see. Even as some vulnerabilities are being weeded out, the software may be collecting a cargo of other vulnerabilities that are invisible to the analyzer. Therefore, a security scanner should not be an organization’s only means of judging the quality of their software. Otherwise, the software might *appear* to be improving when in reality it is staying about the same or actually getting worse.

Return on Investment

Most organizations find that false alarms are a serious obstacle to the satisfactory use of security analyzers. The problem is that in a large software system, it is expensive to go through the analyzer’s output and separate the false alarms from the real vulnerabilities. On the other hand, it can also be expensive to fix every potential security bug regardless of whether it is a false alarm. In many software systems, modularity and efficiency considerations also make it undesirable to fix every unexploitable security bug.

False alarms are a particularly serious problem with the most popular freeware security analyzers. With a handful of exceptions, no work has been done on these analyzers since about 2001, and their technology is considered obsolete by most security professionals. Some commercial tool vendors are actively working to develop technology that reduces false alarms, with varying degrees of success. When evaluating a security analyzer, an organization should be aware of how many false alarms they can expect and whether it will be cost-effective to deal with those false alarms.

In spite of these considerations, most security analyzers provide built-in documentation that says why something is a potential security bug and how to fix it. This is true even for the older freeware analyzers. Therefore, these analyzers can still be a learning tool for software developers.

False alarms are less of a problem when a security analyzer is used to check coding style. Organizations that use a security analyzer in this way are usually willing to put in the extra time needed for highly defensive programming, and making the software extremely efficient might not be their biggest priority either. One way to get an idea of the effort needed for this is to enable all compiler warnings and try to write code that generates no warnings when compiled in this way.

It bears emphasizing that when a security analyzer is used as a style checker, it will point out many issues that do not lead to true vulnerabilities. The thing to keep in mind is that security analyzers examine software at a very low level, so they cannot determine whether a potential vulnerability is forestalled at the design or architecture level. In a sense, security style checkers insist that certain security requirements be enforced at a highly granular level in the source code, even if the software design calls for those requirements to be implemented in some other way.

When using a security analyzer as a badness-ometer, it must be kept in mind that there is no actual science behind the practice of badness-ometering, nor do tool vendors actively strive to make their tools better in this role. Therefore, no real basis exists for preferring one tool over another as a badness-ometer. It may not be cost-effective to invest in an expensive commercial tool if it will be used only for this purpose.

References

[NIST 02]	NIST. *The Economic Impacts of Inadequate Infrastructure for Software Testing* (Planning Report 02-3). Gaithersburg, MD: National Institute of Standards and Technology, 2002. http://www.nist.gov/director/prog-ofc/report02-3.pdf.
[McGraw 04]	McGraw, Gary. "Application Security Testing Tools: Worth the Money?" *Network Magazine,* November 1, 2004. http://www.networkmagazine.com/showArticle.jhtml?articleID= (2004).

Cigital, Inc. Copyright

1. mailto:copyright@cigital.com

Fields

Name	Value
Copyright Holder	Cigital, Inc.

Fields

Name	Value
is-content-area-overview	false
Content Areas	Tools/Code Analysis
Workflow State	Publishable